**Chapter 9 Notes – Composite Pattern**

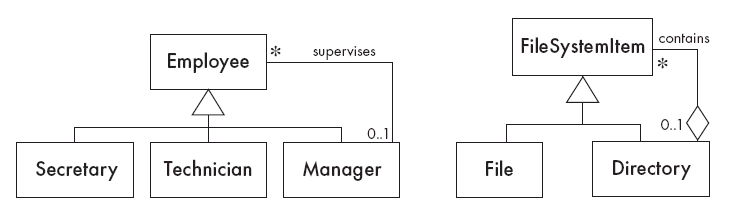
**Definition & Use**

1. “The Composite Pattern allows you to compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.”[HFDP, 356][GOF, 163]
2. “The Composite pattern allows you to build complex objects by recursively composing similar object in a tree-like manner. The Composite pattern also allows the object in the tree to be manipulated in a consistent manner, by requiring all of the objects in the tree to have a common interface or superclass.” [PIJ-1, 187]
3. Use the Composite pattern when:
4. You have a complex object that can be broken down into a part-whole hierarchy of smaller objects.
5. The majority of the relationships do not need to be distinguished. In other words, mostly, the client doesn’t need to know the concrete classes and all manipulation can be carried out through the interface.
6. General UML, two choices (a third choice is shown later):

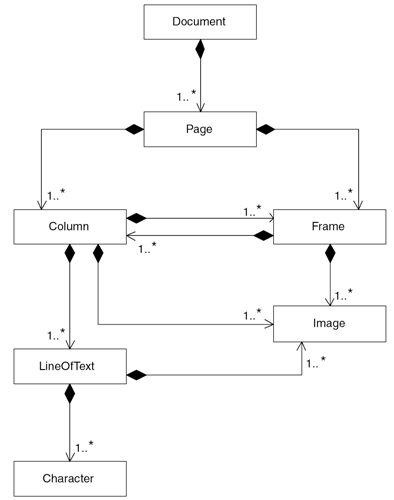
|  |  |  |
| --- | --- | --- |
| Book’s Approach |  | Another Approach |
| E:\Data-Classes\CS 4322 - Software Engineering 2\Notes\09-Iterator&Composite\a3.jpg |  | E:\Data-Classes\CS 4322 - Software Engineering 2\Notes\09-Iterator&Composite\a2.jpg |

**Examples**

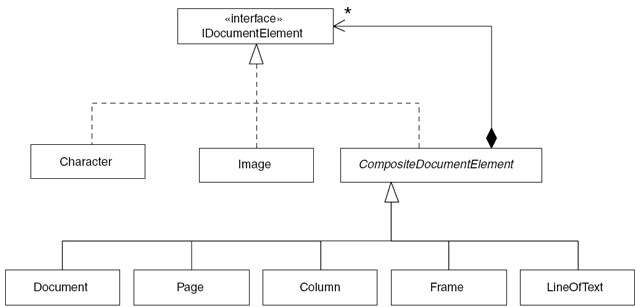
1. File systems are organized with the Composite pattern. HTML elements are also.



1. Consider the organization of a document composed of a number of components.

****

This can be organized using the Composite pattern.

****

**Text Example**

1. How does a menu get its list of components? There are three design options and three factors to consider:
2. Flexibility – Can we modify the menu?
3. Transparency – Do we have to know concrete classes?
4. Safety – Can unsupported operations be called?

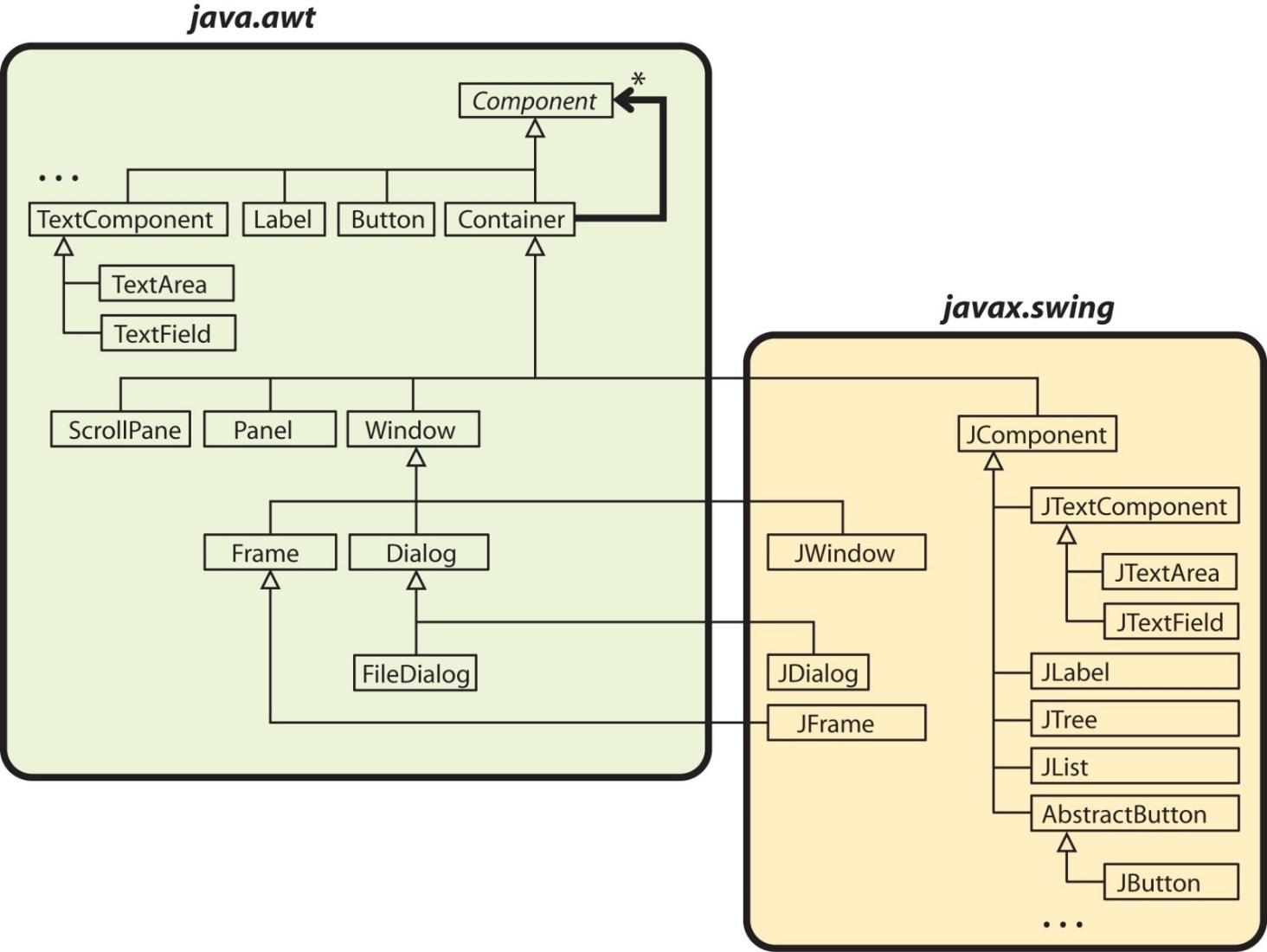
|  |  |  |  |
| --- | --- | --- | --- |
| **Approach** | **1** | **2** | **3** |
| **Structure** | E:\Data-Classes\CS 4322 - Software Engineering 2\Notes\09-Iterator&Composite\pics\bb1.jpg | E:\Data-Classes\CS 4322 - Software Engineering 2\Notes\09-Iterator&Composite\pics\bb2.jpg | E:\Data-Classes\CS 4322 - Software Engineering 2\Notes\09-Iterator&Composite\pics\bb3.jpg |
| **Flexibility** | 🗶 Menu’s cannot be changed | ✓ Menus can be changed | ✓ Menus can be changed |
| **Transparency** | ✓ All items can be referred to as MenuComponents | 🗶 Must know class | ✓ All items can be referred to as MenuComponents |
| **Safety** | ✓ | ✓ | 🗶 It is possible to call the add method on a MenuItem |

1. These three options demonstrate that a composite often has to decide between flexibility, transparency and safety when components are created. You can get any two of the three properties, but not all three.

For some reason, the text only mentions the third technique. However, this is not the common choice.

**Java AWT Package**

1. The designers of the Java AWT classes chose the second technique. The *add* method is in the *Container* class. Thus, to add a *Component* to a *Container*, you must know that the *Container* is a *Container*. So, you can’t add a *Button* to a *Button* (safety?). However, in Swing, a *JButton* is a *Container*, so you can add a *JButton* to a *JButton* (see next example).



1. As mentioned above, Swing makes all controls *Container*s. This design allows us create recursive menus and buttons with embedded buttons and radio buttons.

|  |  |
| --- | --- |
| **E:\Data-Classes\CS 4322 - Software Engineering 2\Notes\09-Iterator&Composite\pics\aa3.jpg** |  |

1. **– JavaFX – The Scene Class Hierarchy**
2. The figure below shows the framework for building a JavaFX GUI, which utilizes the second technique from above. Note that a *Scene* can contain a *Pane*, but in general, the association is more general; the *Scene* has an association with the *Parent* class of which *Pane* is a subclass. Note, also that a *Pane* can have many *Node* instances where each *Node* instance can be a GUI Control, another *Pane*, or other items such as a *MediaView* object.

